Propozycje zadań projektowych

1. Profil Ravenscar Ada – metoda wytwarzania aplikacji wysoko zintegrowanych
2. Tworzenie aplikacji rozproszonych z zastosowaniem języka Ada.
3. Wprowadzenie do programowania generycznego w języku Ada.
4. Interfejs RTAI – specyfikacja, narzędzia programowe.
5. Mechanizmy i narzędzia programowania systemów wbudowanych na platformę WindowsXP Embedded.
6. Praktyczna realizacja przerwań Ada95/83 w systemie GNAT-Linux.
7. Programowanie współbieżne i rozproszone w języku Java.
8. Techniki programowania współbieżnego i rozproszonego na platformę Windows.
9. Wątki czasu rzeczywistego w Real-Time Java.
10. Biblioteki kryptograficzne na platformę Java.
11. Biblioteki kryptograficzne na platformę Linux.

Dostęp do rejestrów urządzeń

- **Ada posiada pełny zestaw mechanizmów do specyfikowania implementacji typów danych.**
- **Mechanizmy nazywane są klauzulami reprezentacji (ang. Representation clauses) i wskazują jak typy języka zostaną zamapowane we wskazanym sprzęcie.**
- **Dostępne są następujące klauzule:**
    - **Klauzule definicji atrybutów: umożliwiają powiązanie atrybutów z zadaniami, obiektami chronionymi lub podprogramami: np. rozmiar obiektu, uporządkowanie pamięci, maksymalna ilość pamięci przeznaczona na zadanie, adres obiektu;**
    - **Klauzule wyliczeniowe: literały i typy wyliczeniowe mogą uzyskać ustaloną wartość**
    - **Klauzule reprezentacji rekordów: dla składników rekordów można zdefiniować przesunięcie i długość w obrębie danej jednostki pamięci;**
    - **Klauzula adresu (Ada83) – przestarzałe – możliwość ustawienia obiektu w określonym adresie.**

Rozważany przykład – definicja rejestru sterującego urządzenia

```
type Error_T is (Read_Error, Write_Error,
                 Power_Fail, Other);
type Function_T is (Read, Write, Seek);
type Unit_t is new Integer range 0 .. 7;

type Csr_T is record
  Errors      : Error_T;
  Busy        : Boolean;
  Unit        : Unit_T;
  Done        : Boolean;
  Ienable     : Boolean;
  Dfun        : Function_T;
  Denable     : Boolean;
end record;
```

Rejestr urządzenia widoczny w postaci rekordu zdefiniowanego przez programistę

Klauzule wyliczeniowe

- **Klauzule wyliczeniowe umożliwiają zdefiniowanie wewnętrznych wartości liczbowych dla literałów:**
  01 – Read
  10 – Write
  11 - Seek

```
type Function_T is (Read, Write, Seek);
for Function_T use (Read=>1,Write=>2,Seek=>3);
```

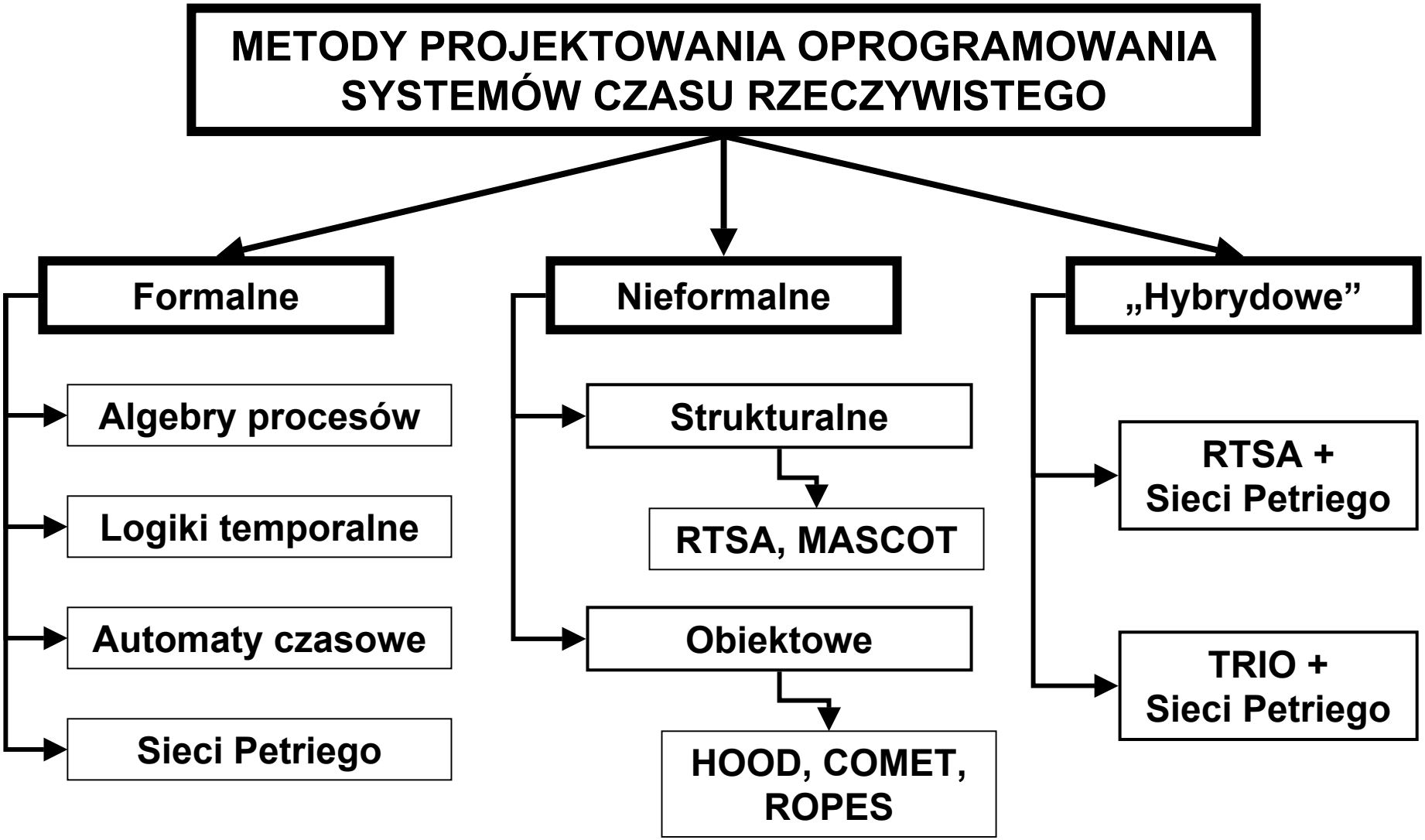
Klauzule reprezentacji rekordów

- **Klauzule reprezentacji rekordów specyfikują sposób organizacji rekordów w pamięci, tj. kolejność pól, pozycję i ich rozmiar.**
- **Bity w rekordach są numerowane od 0, zasięg (ang. range) specyfikuje ilość bitów, które zostaną zaalokowane.**
- **Można również sformułować rozmiar, ustawienie i kolejność bitów.**

```
Word : constant :=2; --ilość bajtów w słowie
Bits_In_Word : constant := 16; -- ilość bitów w słowie
for Csr_T use
  record
    Denable  at 0*Word range 0..0;
    Dfun     at 0*Word range 1..2;
    Ienable  at 0*Word range 6..6;
    Done     at 0*Word range 7..7;
    Unit     at 0*Word range 8 .. 10;
    Busy     at 0*Word range 11 .. 11;
    Errors   at 0*Word range 12 .. 15;
  end record;
for Csr_T'Size use Bits_In_Word;
for Csr_T'Alignment use Word;
for Csr_T'Bit_Order use Low_Order_First;
```

Zdefiniowanie i zastosowanie dostępu do rejestru

```
Tcsr : Csr_T;
for Tcsr'Address use
System.Storage_Elements.To_Address( 8#177566#);
Tmp :Csr_T;

-- The hardware register can be manipulated:
Tmp := (Denable => True, Dfun => Read,
        Ienable => True, Done => False,
        Unit => 4, Errors => None );
Tcsr := Tmp; -- to ensure all bits are set at once

-- To test for errors
if Tcsr.Error = Read_Error then
  raise Disk_Error;
end if;
```

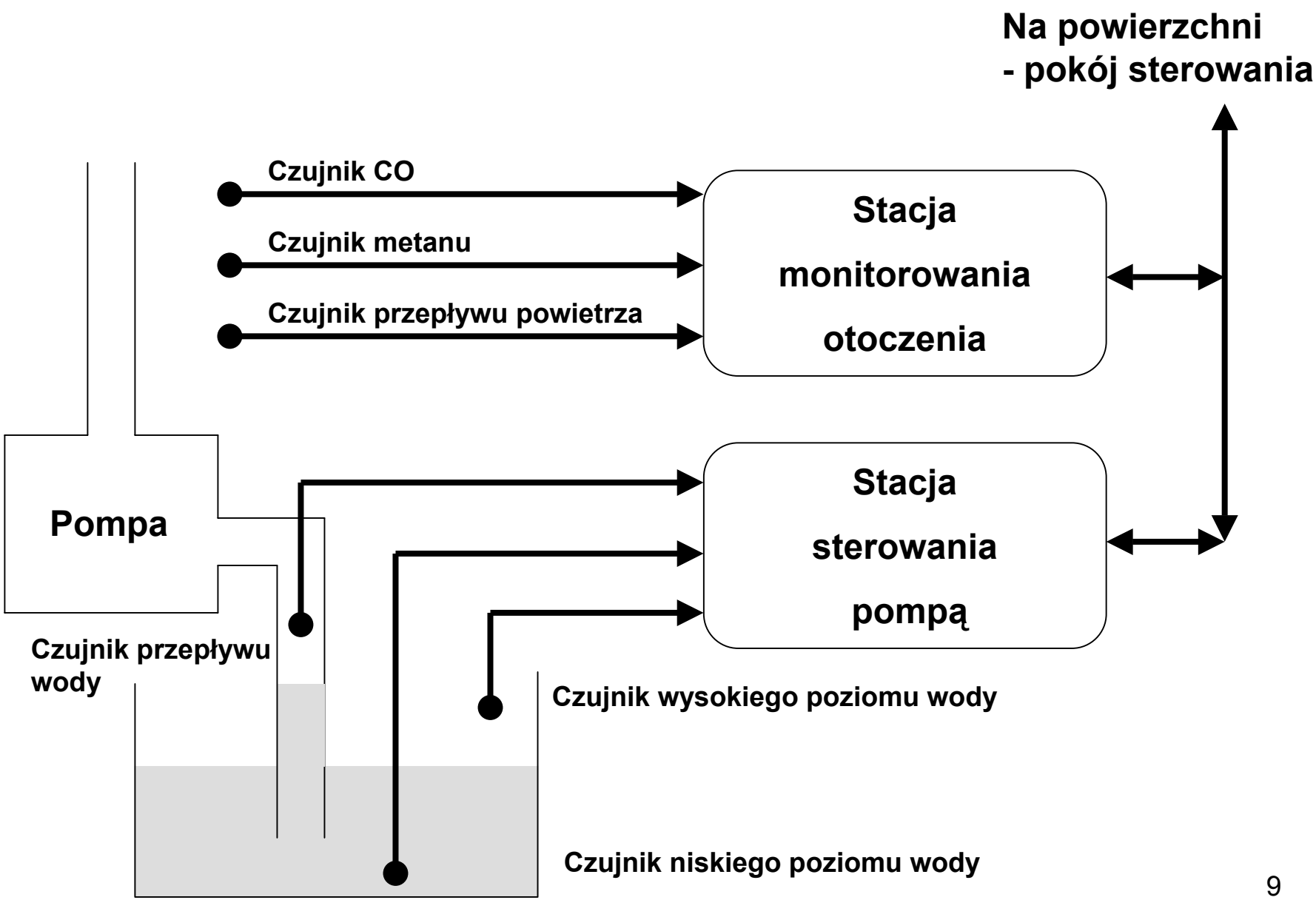

METODY PROJEKTOWANIA OPROGRAMOWANIA SYSTEMÓW CZASU RZECZYWISTEGO
Formalne
Algebry procesów
Logiki temporalne
Automaty czasowe
Sieci Petriego
Nieformalne
Strukturalne
RTSA, MASCOT
Obiektowe
HOOD, COMET, ROPES
„Hybrydowe”
RTSA + Sieci Petriego
TRIO + Sieci Petriego

Porównanie formalnych i nieformalnych metod projektowania

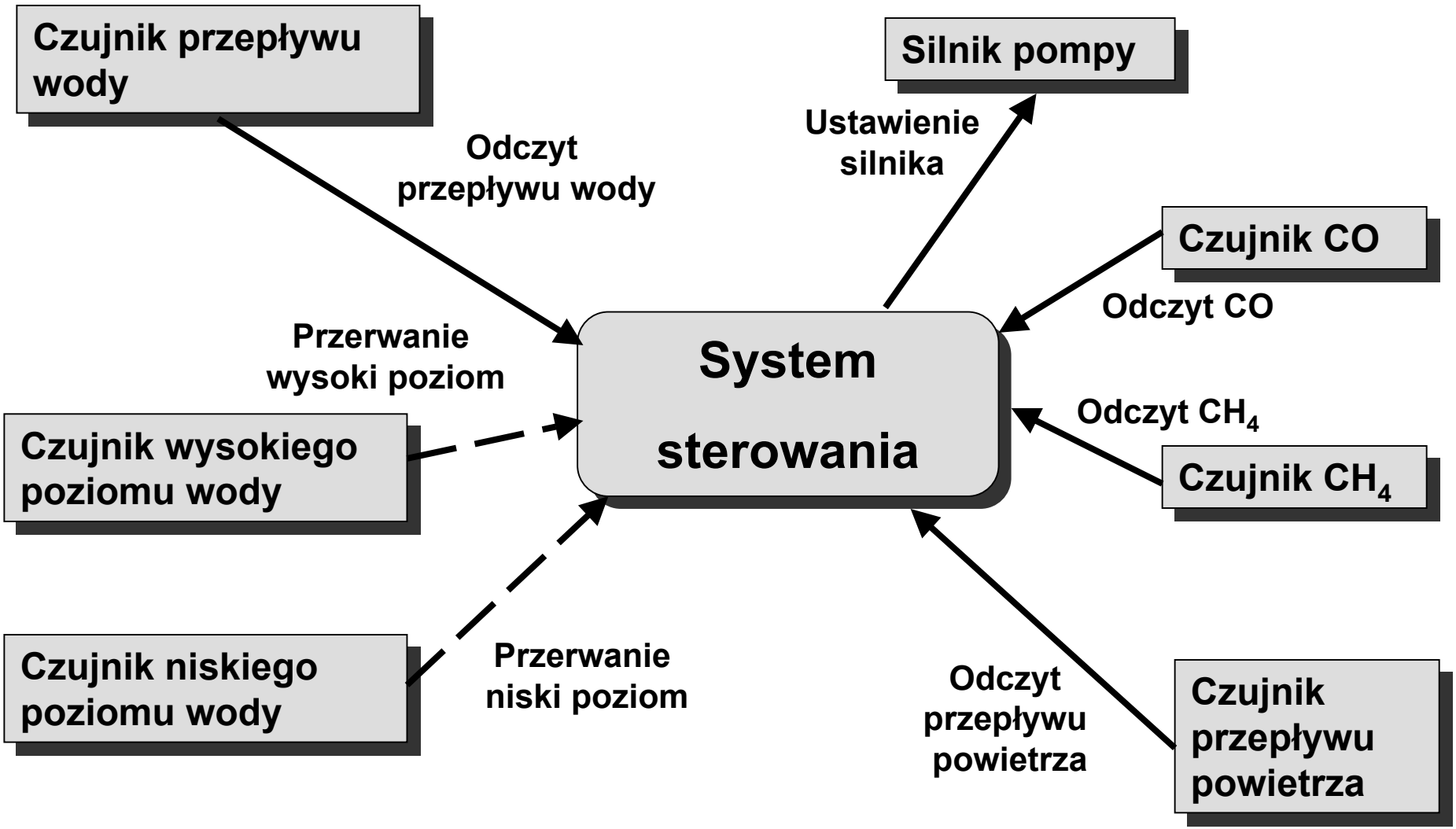
- **Metody nieformalne:**
  - **Reprezentacja systemu z wielu perspektyw**
  - **Systematyczny opis systemu na kolejnych etapach wytwarzania**
  - **Stosowanie wzorców projektowych**

- **Metody formalne:**
  - **Ścisły, matematyczny opis systemu**
  - **Możliwość formalnego wykazywania określonych własności**
  - **Mniejsza siła wyrazu w porównaniu z metodami formalnymi**
  - **Stosowanie zwykle we wczesnych etapach wytwarzania, lub do opisu systemu na wysokim poziomie abstrakcji**

Przykład – system odwadniania kopalni - specyfikacja



Przykład – urządzenia zewnętrzne

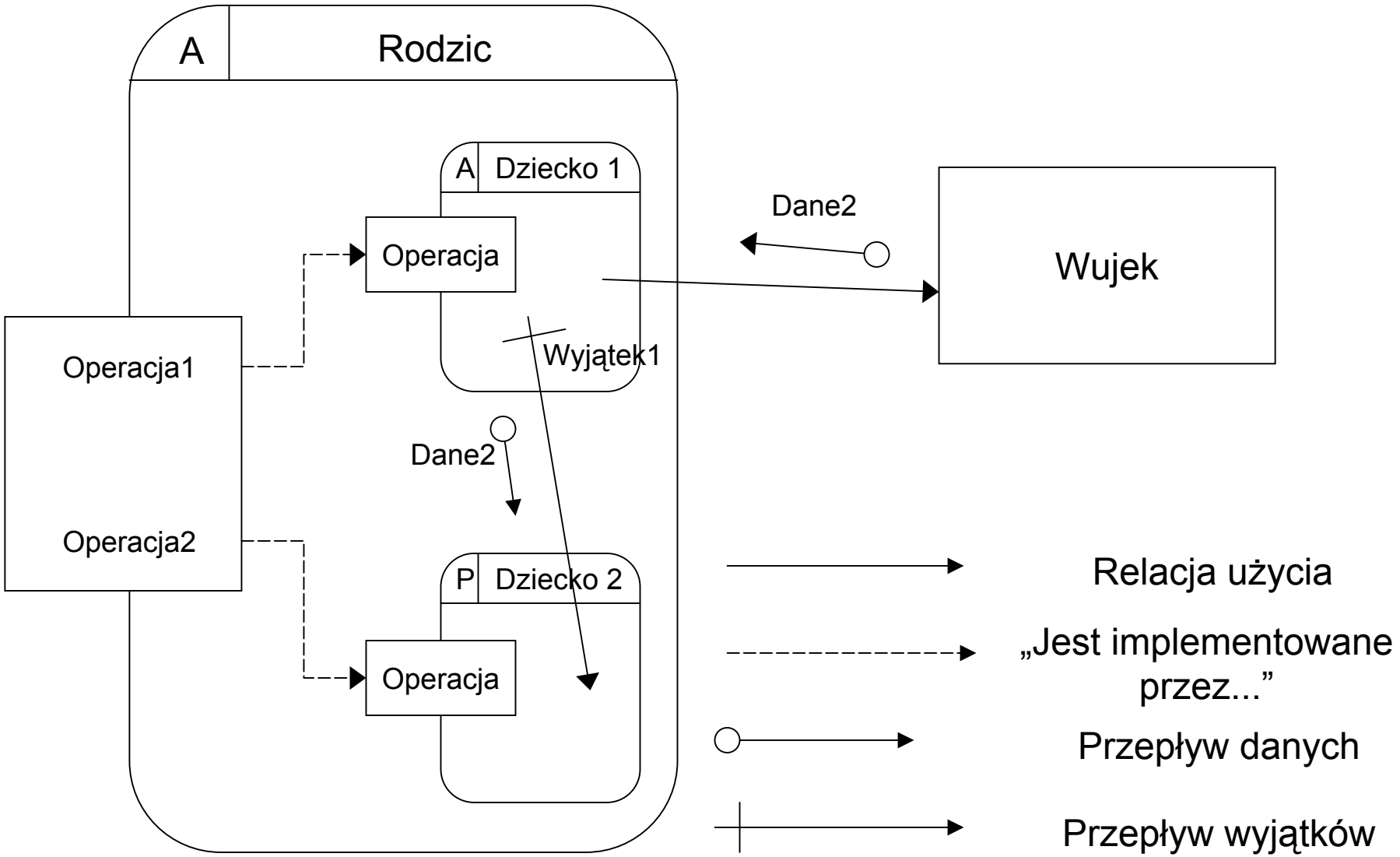


Przykład –ograniczenia czasowe

	Typ	Okres	WCET	Deadline	Priorytet
Czujnik CH_4	Cykliczny	80	12	30	10
Czujnik CO	Cykliczny	100	10	60	8
Czujnik przepływu powietrza	Cykliczny	100	10	100	7
Czujnik przepływu wody	Cykliczny	1000	10	40	9
Procedura obsł. przerwania	Sporadyczny	6000	20	200	6
Silnik	Chroniony				10
Sterownik przerwania	Chroniony				11
Konsola operatora	Chroniony				10
Log	Chroniony				10
Obsługa przerwania	Sporadyczny		2		

	Czas
Okres zegara	20
Narzut zegara	2
Koszt przesunięcia zadania	1

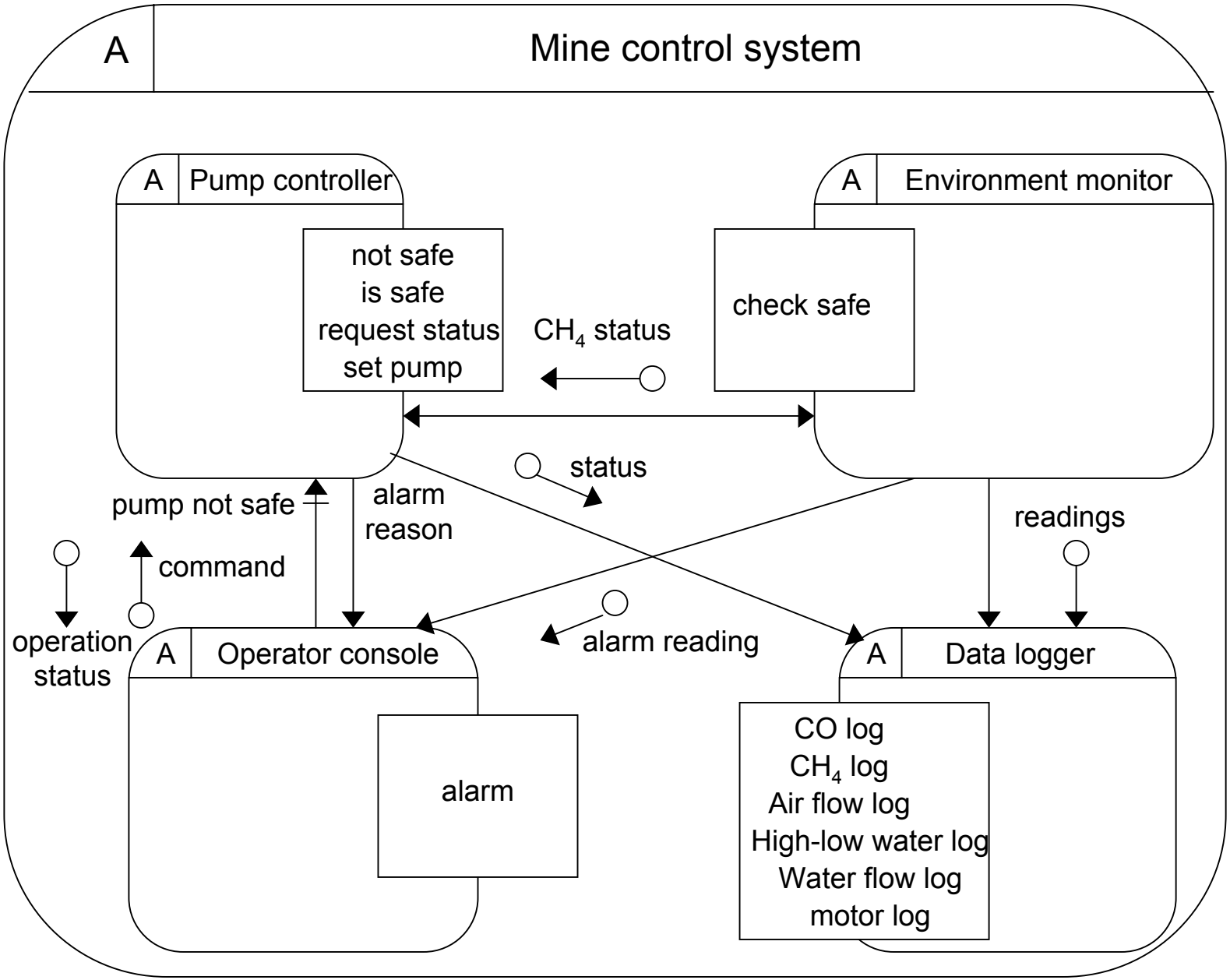
Przykład – notacja HRT-HOOD



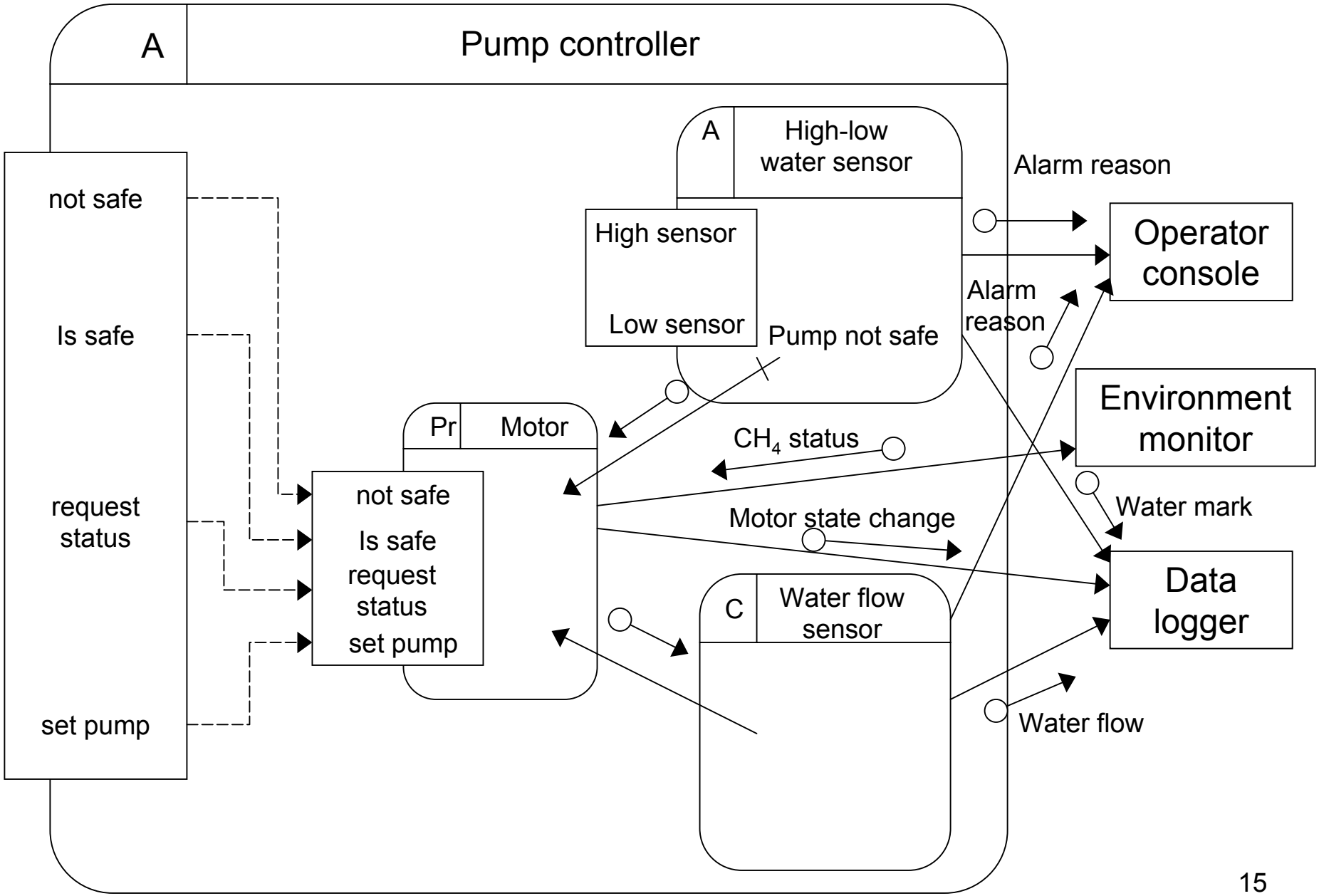
Przykład – notacja HRT-HOOD

- **Typy obiektów w metodyce HRT-HOOD:**
  - **Pasywne – współużywane obiekty nie posiadające mechanizmów kontrolnych nad wywoływanymi w nich operacjach i nie wywołujące spontanicznie operacji w innych obiektach.**
  - **Aktywne – obiekty mogące kontrolować wywołania ich operacji i mogące spontanicznie wywoływać operacje w innych obiektach (najbardziej generalna klasa obiektów).**
  - **Chronione – obiektu, które mogą zawierać kontrolę nad wywoływanymi w nich operacjami i nie wywołują spontanicznie operacji w innych obiektach. W ogólnym przypadku obiekty chronione nie muszą posiadać z góry założonych ograniczeń synchronizacyjnych i muszą być analizowalne jeśli chodzi o czasy zablokowań, które powodują w wykonywaniu swoich klientów.**
  - **Cykliczne – obiekty reprezentujące okresowe wykonywanie obliczeń, mogące spontanicznie wołać inne obiekty z ograniczeniami na interfejs**
  - **Sporadyczne – obiekty reprezentując reakcje na sporadyczne zdarzenia w systemie, mogące spontanicznie wołać operacje w innych obiektach. Każdy obiekt sporadyczny posiada jedną operację sporadyczną.**
- **Obiekty aktywne ostatecznie muszą zostać zdekomponowane na zbiór obiektów chronionych/cyklicznych/sporadycznych/pasywnych.**

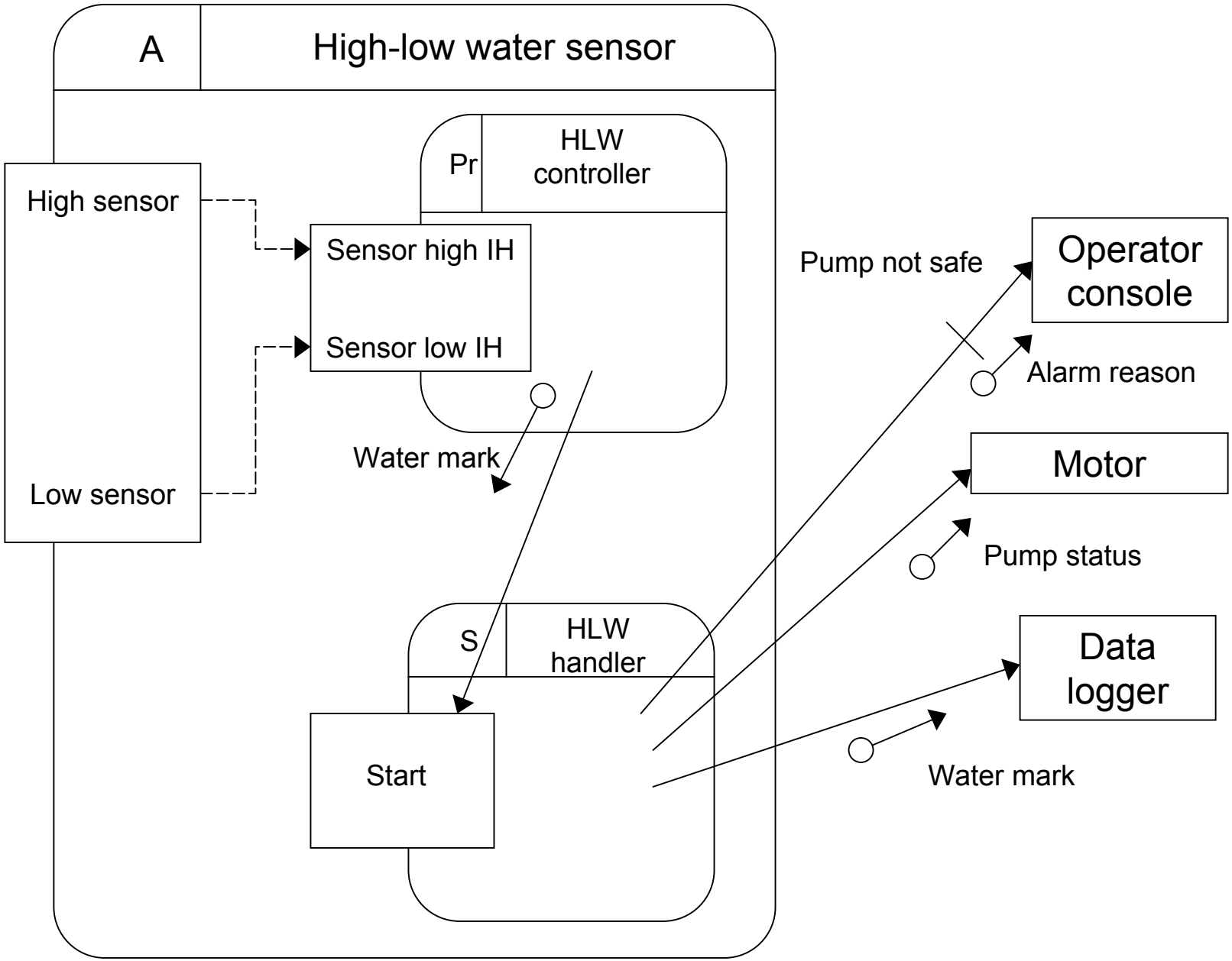
Przykład – pierwszy poziom dekompozycji



Przykład – hierarchiczna dekompozycja obiektu Pump controller



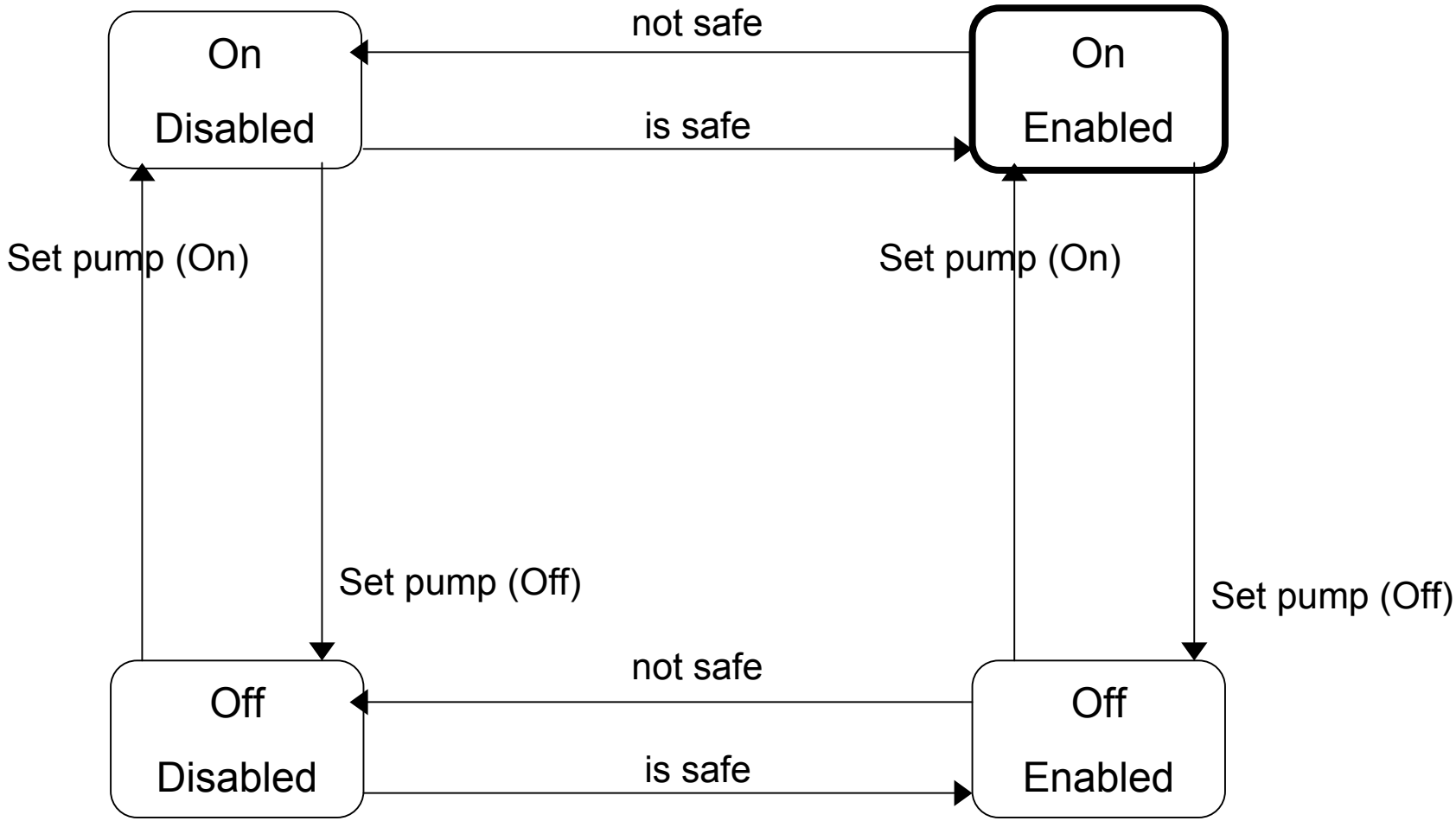
Przykład – hierarchiczna dekompozycja obiektu high-low water sensor



Przykład – specyfikacja obiektu motor

```
-- atrybuty czasu rzeczywistego:
package motor_RTATT is
   ceiling_priority: constant := 6;
end motor_RTATT;
-- interfejs obiektu motor:
package motor is -- PROTECTED

   type pump_status is (on, off);
   type pump_condition is (enabled, disabled);

   type motor_state_changes is (motor_started,
        motor_stopped, motor_safe, motor_unsafe);
   PUMP_NOT_SAFE : exception;

   procedure not_safe;
   procedure is_safe;

   function request_status return pump_status;
   procedure set_pump(to : pump_status); end motor;
```

Przykład – diagram stanów dla obiektu motor



Przykład – definicja dostępu do rejestrów sterujących silnika

```
package device_register_types is

  word : constant := 2;       -- two bytes in a word
  one_word : constant := 16; -- 16 bits in a word
  -- register field types
  type device_error is (clear, set);
  type device_operation is (clear, set);
  type interrupt_status is (I_disabled, I_enabled);
  type device_status is (D_disabled, D_enabled);

  -- register type itself
  type csr is
    record
      error_bit   : device_error;
      operation   : device_operation;
      done        : boolean;
      interrupt   : interrupt_status;
      device      : device_status;
    end record;
```

Przykład – definicja dostępu do rejestrów sterujących silnika

```
-- bit representation of the register field
  for device_error use (clear => 0, set => 1);
  for device_operation use (clear => 0, set => 1);
  for interrupt_status use (I_disabled => 0,
                            I_enabled => 1);
  for device_status use (D_disabled => 0,
                         D_enabled => 1);
  for csr use
    record at mod word;
      error_bit  at 0 range 15 ..15;
      operation  at 0 range 10 ..10;
      done       at 0 range 7 .. 7;
      interrupt  at 0 range 6 .. 6;
      device     at 0 range 0 .. 0;
    end record;
  for csr'size use one_word;
end device_register_types;
```

Przykład – implementacja obiektu motor

```
with CH4_status; use  CH4_status;
with device_register_types; use device_register_types;
with system; use system;
with motor_rtatt; use motor_rtatt;
package body motor is

    -- define control and status register
    -- for the motor
  --Control_reg_addr : constant Address := 16#AA14#;
  pcsr : device_register_types.csr :=
      (error_bit => clear, operation => set,
       done => false, interrupt => I_enabled,
       device => D_enabled);
  --for pcsr'address use Control_reg_addr;
```

Przykład – implementacja obiektu motor

```
protected Agent is
  PRAGMA PRIORITY(motor_rtatt.
                  ceiling_priority);
  procedure not_safe;
  procedure is_safe;
  function request_status return pump_status;
  procedure set_pump(to : pump_status);
private
  motor_status : pump_status := off;
  motor_condition : pump_condition := enabled;
end Agent;
```

Przykład – implementacja obiektu motor

```
procedure not_safe is
begin
  Agent.not_safe;
end not_safe;

procedure is_safe is
begin
  Agent.is_safe;
end is_safe;

function request_status return pump_status is
begin
    return Agent.request_status;
end request_status;

procedure set_pump(to : pump_status) is
begin
    Agent.set_pump(to);
end set_pump;
```

Przykład – implementacja obiektu motor

```
protected body Agent is
  procedure not_safe is
  begin
    if motor_status = on then
      pcsr.operation := clear; -- turn off motor
    end if;
    motor_condition := disabled;
    data_logger.motor_log(motor_unsafe);
  end not_safe;

  procedure is_safe is
  begin
    if motor_status = on then
      pcsr.operation := set; -- start motor
      data_logger.motor_log(motor_started);
      motor_condition := enabled;
    end if;
    data_logger.motor_log(motor_safe);
  end is_safe;
```

Przykład – implementacja obiektu motor

```
function request_status return pump_status is
begin
  return motor_status;
end request_status;
```

Przykład – implementacja obiektu motor

```
procedure set_pump(to : pump_status) is
    begin
      if to = on then
       if motor_status = off or motor_condition = disabled then
          if CH4_status.read  = motor_safe then
            -- Environment_Monitor.Check_Safe
            motor_status := on;
            motor_condition := enabled;
            pcsr.operation := set; -- turn on motor
            data_logger.motor_log(motor_started);
          elsif motor_condition = disabled then
            raise pump_NOT_SAFE;
        end if;  end if;
      else
        if motor_status = on then
          pcsr.operation := clear; -- turn off motor
          motor_status:= off;
          data_logger.motor_log(motor_stopped);
        end if; end if; end set_pump; end Agent; end motor;
```

Przykład – specyfikacja obiektu waterflow sensor

```
-- ustawienia czasu rzeczywistego:
with system; use system;
with Ada.Real_time; use Ada.Real_time;
package water_flow_sensor_RTATT is

  period : Time_Span :=  To_Time_Span(60.0);
  thread_priority : constant priority := 2;

end water_flow_sensor_RTATT;

-- specyfikacja pakietu:
package water_flow_sensor is -- CYCLIC
  pragma elaborate_body;
  type water_flow is (yes, no);
                     -- calls Operator_Console.Alarm
                     -- calls Data_Logger.Water_Flow_Log
                     -- calls Motor.Request_Status
end water_flow_sensor;
```

Przykład – implementacja obiektu waterflow sensor

```
with Ada.Real_Time; use Ada.Real_Time;
with device_register_types; use device_register_types;
with system; use system;
with water_flow_sensor_rtatt; use water_flow_sensor_rtatt;
with motor; use motor;
with operator_console; use operator_console;
with data_logger; use data_logger;
package body water_flow_sensor is

  flow : Water_flow := No;
  current_pump_status, last_pump_status : pump_status := off;

  -- define control and status register
  -- for the flow switch
    -- define control and status register
    -- for the motor
  -- Control_reg_addr : constant Address := 16#AA14#;
  wfcsr : device_register_types.csr;
  -- for wfcsr'address use Control_reg_addr;
```

Przykład – implementacja obiektu waterflow sensor

```
procedure initialise is
 begin  -- enable device
   wfcsr.device := D_enabled;
 end initialise;

 procedure PERIODIC_CODE is  begin
   current_pump_status := motor.request_status;
   if (wfcsr.operation = set) then   flow :=  yes;
   else                               flow := no;
   end if;
   if current_pump_status = on and
       last_pump_status = on and flow = no  then
     operator_console.alarm(PUMP_DEAD);
   elsif current_pump_status = off and
       last_pump_status = off and flow = yes  then
     operator_console.alarm(PUMP_DEAD);
   end if;
   last_pump_status := current_pump_status;
   data_logger.water_flow_log(flow);
 end PERIODIC_CODE;
```

Przykład – implementacja obiektu waterflow sensor

```
  task THREAD is
    pragma priority(water_flow_sensor_rtatt.thread_priority);
  end thread;

  task body THREAD is
    T: TIME;
    PERIOD : time_span := water_flow_sensor_rtatt.period;
  begin
    T:= clock;
    initialise;
    loop
      periodic_code;
      T := T + PERIOD;
      delay until(T);
    end loop;
  end THREAD;

end water_flow_sensor;
```

Przykład – specyfikacja obiektu HLW controller

```
-- specyfikacja ograniczeń czasu rzeczywistego:
with system; use system;
with Ada.Real_time; use Ada.Real_time;
package HLW_controller_RTATT is
   -- for PROTECTED objects
   ceiling_priority :  constant priority :=  9;
end HLW_controller_RTATT;

-- specyfikacja pakietu zawierającego procedury obsługi
-- przerwań:
with HLW_controller_rtatt; use HLW_controller_rtatt;
package HLW_controller is -- PROTECTED
   procedure sensor_high_IH;
   procedure sensor_low_IH;
end HLW_controller;
```

Przykład – implementacja obiektu HLW controller

```
with HLW_handler; use HLW_handler;
with system; use system;
-- with Ada.Interrupts; use Ada.Interrupts;
-- with Ada.Interrupts.Names; use Ada.Interrupts.Names;
package body HLW_controller is
  protected Agent is
    PRAGMA PRIORITY(HLW_controller_rtatt.
                ceiling_priority);
    procedure sensor_high_IH;
    --pragma Attach_Handler(sensor_high_IH, WaterH_Interrupt);
          -- assigns interrupt handler
    procedure sensor_low_IH;
    --pragma Attach_handler(sensor_low_IH, WaterL_Interrupt);
          -- assigns interrupt handler
  private
  end Agent;
```

Przykład – implementacja obiektu HLW controller

```
procedure sensor_high_IH is
  begin     Agent.sensor_high_IH;
  end  sensor_high_IH;

 procedure sensor_low_IH is
  begin    Agent.sensor_low_IH;
  end sensor_low_IH;

 protected body Agent is
    procedure sensor_high_IH is
    begin      HLW_handler.START(high);
    end sensor_high_IH;

    procedure sensor_low_IH is
    begin      HLW_handler.START(low);
    end sensor_low_IH;
  end Agent;
end HLW_controller;
```

Przykład – specyfikacja obiektu HLW handler

```
-- ograniczenia czasu rzeczywistego:
with system; use system;
with Ada.Real_time; use Ada.Real_time;
package HLW_handler_RTATT is
  ceiling_priority :  constant priority := 11;
  thread_priority :  constant priority := 6;
  MAT  :  Time_Span := To_Time_Span(100.0);
end HLW_handler_RTATT;

-- sporadyczny obiekt zwracający stan czujników poziomu wody
with HLW_handler_rtatt; use HLW_handler_rtatt;
with Ada.Real_Time; use Ada.Real_Time;
package HLW_handler is -- SPORADIC
  type water_mark is (high, low);
  procedure START(int : water_mark);
end HLW_handler;
```

Przykład – implementacja obiektu HLW handler

```
with device_register_types; use device_register_types;
with system; use system;
with motor; use motor;
with data_logger;
package body HLW_handler is

   -- define control and status registers
   -- for the high and low water switches
   --HW_Control_reg_addr : constant Address := 16#AA10#
   --LW_Control_reg_addr : constant Address := 16#AA12#
   hwcsr : device_register_types.csr;
   --for hwcsraddress use HW_Control_reg_addr;
   lwcsr : device_register_types.csr;
   --for lwcsr'address use LW_Control_reg_addr;
   -- procedure initialise is separate;
   -- procedure sporadic_code(int : water_mark) is separate;
```

Przykład – implementacja obsługi przerwań (HLW handler)

```
procedure sporadic_code(int : water_mark) is
begin
  if int = high then
    motor.set_pump(on);
    data_logger.high_low_water_log(high);
  else
    motor.set_pump(off);
    data_logger.high_low_water_log(low);
  end if;
end sporadic_code;
```

Przykład – implementacja obsługi przerwań (HLW handler)

```
procedure initialise is
begin
  hwcsr.device :=D_enabled;
  hwcsr.interrupt := I_enabled;
  lwcsr.device :=D_enabled;
  lwcsr.interrupt := I_enabled;
end initialise;

  task thread is
    pragma PRIORITY(HLW_handler_rtatt.
                    thread_priority);
  end thread;
```

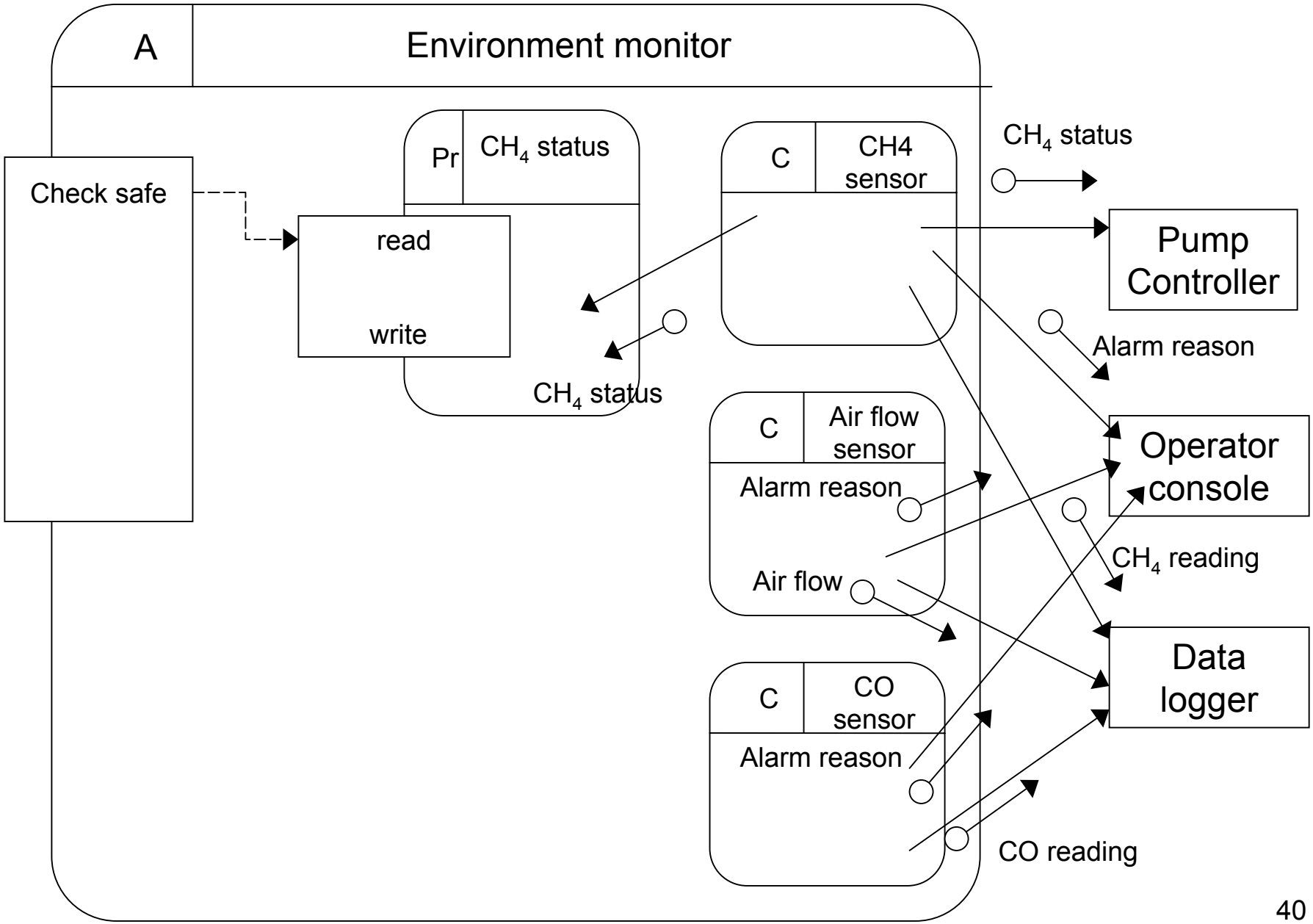
Przykład – implementacja obsługi przerwań (HLW handler)

```
protected Agent is
    pragma PRIORITY(HLW_handler_rtatt.
                    ceiling_priority);
    -- for the Start operation
    procedure START(int : water_mark);
    entry WAIT_START(int : out water_mark);
  private
    START_OPEN : BOOLEAN := FALSE;
    W : water_mark;
  end Agent;

  procedure START(int : water_mark) is
  begin
     Agent.START(int);
  end start;
```

Przykład – implementacja obsługi przerwań (HLW handler)

```
protected body Agent is
  procedure START(int : water_mark) is
    begin       W := int;
                START_OPEN := TRUE;
  end START;
  entry WAIT_START(int : out water_mark)
            when START_OPEN is
            begin       int := W;
                        START_OPEN := FALSE;
            end WAIT_START;
end Agent;

task body thread is
    int : water_mark;
  begin    initialise;
    loop        Agent.WAIT_START(int);
                sporadic_code(int);
    end loop;
  end THREAD; end HLW_handler;
```

Przykład – hierarchiczna dekompozycja obiektu Environment monitor



Przykład – obiekt CH4 status

```
with system; use system;
with Ada.Real_time; use Ada.Real_time;
package ch4_status_RTATT is

   -- for PROTECTED objects
   ceiling_priority :  constant priority :=  7;

end ch4_status_RTATT;

with CH4_status_rtatt; use CH4_status_rtatt;
package CH4_status is -- PROTECTED
   type methane_status is (motor_safe, motor_unsafe);

   function read return methane_status;
   procedure write (current_status : methane_status);
end CH4_status;
```

Przykład – obiekt CH4 status

```
package body CH4_status is
 protected Agent is
    PRAGMA PRIORITY(ch4_status_rtatt.
                    ceiling_priority);
    procedure write (current_status : methane_status);
    function read return methane_status;
  private
    environment_status : methane_status := motor_unsafe;
  end Agent;

  function read return methane_status is
  begin
    return Agent.read;
  end read;

  procedure write (current_status : methane_status)  is
  begin
    Agent.write(current_status);
  end write;
```

Przykład – obiekt CH4 status

```
 protected body Agent is
    procedure write (current_status : methane_status) is
    begin
      environment_status := current_status;
    end write;

    function read return methane_status is
    begin
      return environment_status;
    end read;
  end Agent;

end CH4_status;
```

Przykład – obiekt CH4 sensor handling

```
with system; use system;
with Ada.Real_time; use Ada.Real_time;
package ch4_sensor_RTATT is

   period : Time_Span := To_Time_Span(5.0);
   offset : Time_Span := To_Time_Span(0.0);
   thread_priority : constant priority := 5;

end ch4_sensor_RTATT;

package CH4_sensor is -- CYCLIC
   pragma elaborate_body;
   type CH4_reading is new integer range 0 .. 1023;
   CH4_HIGH : constant CH4_READING := 400;
                       -- calls Motor.Is_Safe
                       -- calls Motor.Not_Safe
                       -- calls Operator_Console.Alarm
                       -- calls Data_Logger.Ch4_Log
end CH4_sensor;
```

Przykład – obiekt CH4 sensor handling

```
with Ada.Real_Time; use Ada.Real_Time;
with system; use system;
with ch4_sensor_rtatt; use ch4_sensor_rtatt;
with device_register_types; use device_register_types;
with operator_console; use operator_console;
with motor; use motor;
with data_logger; use data_logger;
with ch4_status; use ch4_status;
package body CH4_sensor is
  CH4_present : ch4_reading;
  -- define control and status register for the CH4 ADC
  -- Control_reg_addr : constant Address := 16#AA18#
  -- Data_reg_addr : constant Address := 16#AA1A#
  CH4csr : device_register_types.csr;
  --for CH4csr'address use Control_reg_addr;
  -- define the data register
  CH4dbr : ch4_reading;
  --for CH4dbraddress use Data_reg_addr;
  JITTER_RANGE : constant ch4_reading := 40;
  now : TIme;
```

Przykład – obiekt CH4 sensor handling

```
procedure initialise is
begin
   -- enable device
   CH4csr.device := D_enabled;
end initialise;
```

Przykład – obiekt CH4 sensor handling

```
procedure PERIODIC_CODE is
begin
    CH4csr.operation := set; -- start conversion
    now := clock;   -- wait for conversion
    loop exit when clock >= now + Milliseconds(100);
    end loop;
    if not CH4csr.done then
      operator_console.alarm(CH4_DEVICE_ERROR);
    else CH4_present := CH4dbr;
      if CH4_present > CH4_HIGH then
        if CH4_status.read = motor_safe then
          motor.not_safe; CH4_status.write(motor_unsafe);
          operator_console.alarm(high_methane);
        end if;
      elsif (CH4_present < (CH4_HIGH - jitter_range)) and
            (CH4_status.read = motor_unsafe) then
           motor.is_safe; CH4_status.write(motor_safe);
      end if;  data_logger.CH4_log(CH4_present);
    end if;
end PERIODIC_CODE;
```

Przykład – obiekt CH4 sensor handling

```
  task THREAD is
    PRAGMA PRIORITY(ch4_sensor_rtatt.
                    thread_priority);
  end thread;

  task body THREAD is
    T: TIME;
    PERIOD : Time_span := ch4_sensor_rtatt.period;
  begin
    T:= clock;
    initialise;
    loop
      PERIODIC_CODE;
      T := T + PERIOD;
      delay until(T);
    end loop;
  end THREAD;
end CH4_sensor;
```

Przykład – uzupełnienie

- **Opis sytemu należałoby uzupełnić implementacjami obiektów:**
  - **Air flow sensor**
  - **CO sensor**
  - **Data logger**
  - **Operator console**
- **Konstrukcja obiektów Air flow sensor i CO sensor jest podobna do obiektu CH sensor. Nie komunikują się one tylko z obiektem chronionym w celu udostępnienia danych innym składnikom systemu.**
- **Interfejsy do obiektów data logger i operator console zostaną podane na następnych slajdach.**
- **Dokładniejsze studium omawianego przykładu można znaleźć w:**
  - **Burns A., Wellings A.: Real-Time Systems and Programming Languages,Pearson Education Limited 2001.**
  - **Burns A., Wellings A.: HRT-HOOD: A Structured Design Method for Hard Real-Time Ada Systems, Elsevier 1995.**

Przykład – uzupełnienie – interfejs do obiektu data logger

```
with CO_sensor; use CO_sensor;
with CH4_sensor; use CH4_sensor;
with air_flow_sensor; use air_flow_sensor;
with HLW_handler; use HLW_handler;
with water_flow_sensor; use water_flow_sensor;
with motor; use motor;
package data_logger is -- ACTIVE

   procedure CO_log(reading : CO_reading);
   procedure CH4_log(reading : CH4_reading);
   procedure air_flow_log(reading : air_flow_status);
   procedure high_low_water_log(mark : water_mark);
   procedure water_flow_log(reading : water_flow);
   procedure motor_log(state : motor_state_changes);

end data_logger;
```

Przykład – uzupełnienie – interfejs do obiektu operator console

```
package operator_console is -- ACTIVE

  type alarm_reason is (HIGH_METHANE, HIGH_CO, NO_AIR_FLOW,
                        CH4_DEVICE_ERROR, CO_DEVICE_ERROR,
                         PUMP_DEAD, UNKNOWN_ERROR);
  procedure alarm(reason: alarm_reason;
                         Name : String := "Unknown";
                         Details : STRING:= "");

  -- calls Request_Status in Pump_Controller
  -- calls Set_Pump in Pump_Controller

end operator_console;
```